1. **What is Docker?**

   Docker is an open-source containerization platform by which you can pack your application and all its dependencies into a standardized unit called a container.

2. **What is a Container?**

   Containers bundle an application with all its dependencies, libraries, and configuration files, ensuring that it runs consistently across different environments. Unlike virtual machines, containers share the host operating system's kernel, making them more resource-efficient. You can run the docker image as a docker container in any machine where docker is installed without depending on the operating system.

3. **Why we need Docker?**

   Let's consider that you're working on an application which requires multiple technologies to be used for developing the different components like the Frontend, Backend, Database etc… Everything seems to be good, but while you're working with all these technologies on your machine, most often there arises a problem where one technology needs one version of a dependency and some other technology needs a different one to work. Adding to this, it might also happen that your application may not work the same way on someone else's machine or when you deploy it to different environments with different OS or hardware, it fails to run.

   Docker is a tool which helps in developing, shipping, and running your applications on containers and enables you to separate your applications from your infrastructure so you can deliver software quickly. It provides the ability to package and run an application in an isolated environment i.e; containers. The isolation and security allows you to run many containers simultaneously on a given host

4. **What are the Pros and Cons of Docker?**

   **Pros of Docker**

   **Portability:** It enables consistent deployment across various environments.
   Resource Efficiency: Optimizing of resource usage with a shared kernel will be done effectively.
   **Isolation:** It provides security through isolation of process and file system.
   **Automation:** it supports automated builds and streamlining development workflow

   **Cons of Docker**

   **Learning Curve:** Initial learning of the containerization concepts will bit new to understand.
   **Additional Resources:** Containers use some more resources compared to running applications directly on host.
   **Security Concerns:** Misconfigurations may lead to the security risks if not properly managed.
   Container Orchestration Complexity: Management of orchestration tools will be complex for larger-scale deployments.

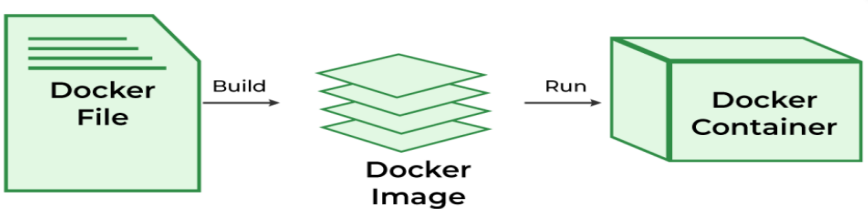
5. **Explain Key Components of Docker?**

**Docker Engine:** It is a core part of docker, that handles the creation and management of containers.
**Docker Image:** It is a read-only template that is used for creating containers, containing the application code and dependencies.
**Docker Hub:** It is a cloud based repository that is used for finding and sharing the container images.
**Dockerfile:** It is a script that containing instructions to build a docker image.
**Docker Registry :** It is a storage distribution system for docker images, where you can store the images in both public and private modes.

6. **Explain about Docker File?**

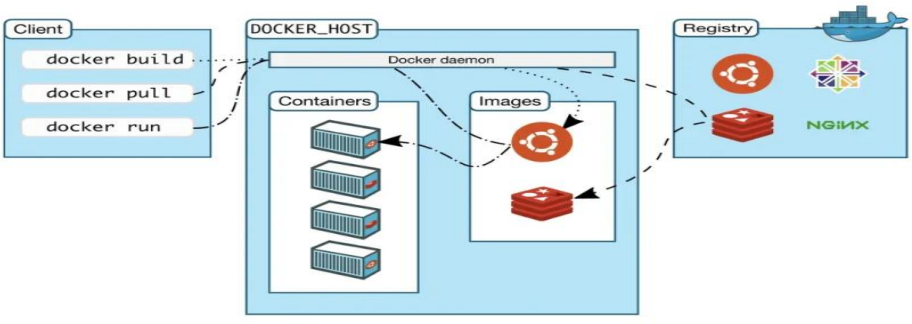
The Dockerfile uses DSL (Domain Specific Language) and contains instructions for generating a Docker image. Dockerfile will define the processes to quickly produce an image. While creating your application, you should create a Dockerfile in order since the Docker daemon runs all of the instructions from top to bottom.

- ✓ It is a text document that contains necessary commands which on execution help assemble a Docker Image.
- ✓ Docker image is created using a Dockerfile.




7. **Explain about Doker Architecture?**

Docker makes use of a client-server architecture. The Docker client talks with the docker daemon which helps in building, running, and distributing the docker containers.




docker architecture

**Docker Engine:**

- ✓ The Docker Engine is the heart of the Docker platform. It comprises two main components:

**Docker Daemon (dockerd):** The Docker daemon runs on the host machine and is responsible for managing Docker objects, such as images, containers, networks, and volumes.
**Docker Client:** The Docker client is a command-line interface (CLI) tool that allows users to interact with the Docker daemon through commands. Users can build, run, stop, and manage Docker containers using the Docker CLI.

**Docker Images:**

- ✓ Docker images are the building blocks of containers. They are read-only templates that contain the application code, runtime, system tools, libraries, and other dependencies. Docker images are created from Dockerfiles, which are text files containing instructions for building the image layer by layer.

**Docker Containers:**

- ✓ Docker containers are runnable instances of Docker images. They encapsulate the application and its dependencies, providing an isolated environment for execution. Containers can be created, started, stopped, moved, and deleted using Docker commands.

**Docker Registry:**

- ✓ Docker Registry is a centralized repository for storing and sharing Docker images. The default public registry is Docker Hub, where users can find a vast collection of images. Organizations can also set up private registries to store proprietary images securely.

**Docker Compose:**

- ✓ Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file (**docker-compose.yml**) to specify services, networks, volumes, and other configurations required for the application. Docker Compose simplifies the management of complex applications composed of multiple interconnected containers.

**Docker Volumes:**

- ✓ Docker volumes are used for persisting data generated by and used by Docker containers. They provide a way for containers to store and share data independently of the container lifecycle, ensuring data persistence and portability.

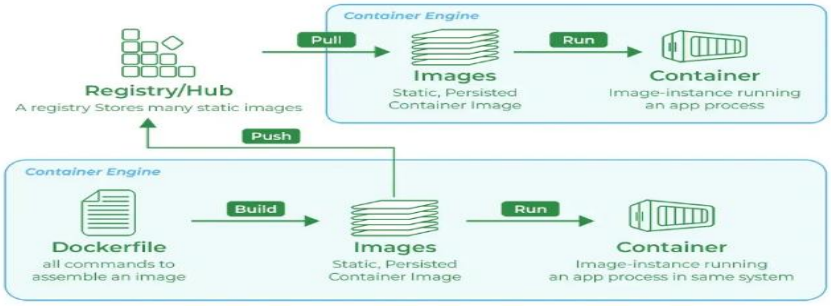
**Docker Networking:**

- ✓ Docker provides networking capabilities for containers to communicate with each other and with external networks. It uses software-defined networks (SDN) to create virtual networks, enabling connectivity and isolation. Users can create custom networks, connect containers to networks, and define network policies using Docker commands or Docker Compose.

**8. Explain about Docker Hub?**

Docker Hub is a repository service and it is a cloud-based service where people push their Docker Container Images and also pull the Docker Container Images from the Docker Hub anytime or anywhere via the internet. Generally it makes it easy to find and reuse images. It

provides features such as you can push your images as private or public registry where you can store and share Docker images.
Mainly DevOps team uses the Docker Hub. It is an open-source tool and freely available for all operating systems. It is like storage where we store the images and pull the images when it is required. When a person wants to push/pull images from the Docker Hub they must have a basic knowledge of Docker. Let us discuss the requirements of the Docker tool.




9. **Explain about steps to be written in Docker File?**

```
# Use the official OpenJDK base image
FROM openjdk:17-jdk-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the built jar file into the container
COPY target/demo-0.0.1-SNAPSHOT.jar app.jar

# Expose port 8080
EXPOSE 8080

# Run the application
ENTRYPOINT ["java", "-jar", "app.jar"]
```

10. **Explain few docker commands?**

**Docker Run:** It used for launching the containers from images, with specifying the runtime options and commands.
**Docker Pull:** It fetches the container images from the container registry like Docker Hub to the local machine.
**Docker ps :** It helps in displaying the running containers along with their important information like container ID, image used and status.
**Docker Stop :** It helps in halting the running containers gracefully shutting down the processes within them.
**Docker Start:** It helps in restarting the stopped containers, resuming their operations from the previous state.

**Docker Login:** It helps to login in to the docker registry enabling the access to private repositories.
**Build Docker Image:** docker build -t demo-app .

- `docker build`: The Docker command to build an image.
- `-t demo-app`: Tags the image with the name `demo-app`.
- `.`: Specifies the current directory as the build context.

**Run Docker Container:** docker run -p 8080:8080 demo-app

- `docker run`: The Docker command to run a container.
- `-p 8080:8080`: Maps port 8080 of the container to port 8080 on the host machine.
- `demo-app`: The name of the Docker image to run.

**List Docker Images:** docker images
**Remove Docker Images:** docker rmi demo-app
**Remove Container:** docker rm my_container

11. **Difference between docker rm IMAGE vs docker rmi IMAGE?**

**docker rm**: This command is used to remove containers. You can specify the name or ID of a container to delete it.
docker rm my-container
**docker rmi**: This command is used to remove images. You specify the name or ID of an image to delete it from your local system.
docker rmi my-image

12. **In Spring Boot how to build the docker image?**

**Step 4: Build the Docker Image**

**4.1 Build the Docker Image**

Run the following command to build the Docker image:

```
docker build -t demo-app .
```

**Explanation:**

- `docker build`: The Docker command to build an image.
- `-t demo-app`: Tags the image with the name `demo-app`.
- `.`: Specifies the current directory as the build context.

13. **In Spring Boot how to run the docker container?**

**5.1 Run the Docker Container**

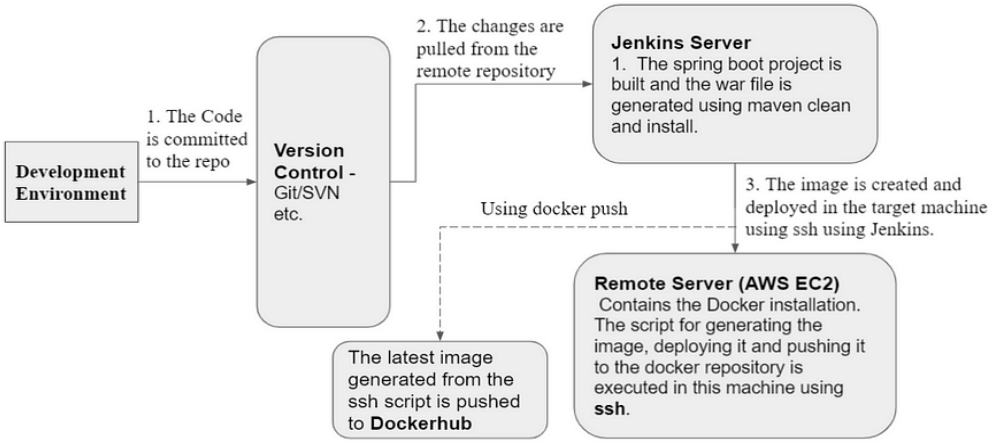
Run the following command to start a Docker container from the image:

```
docker run -p 8080:8080 demo-app
```

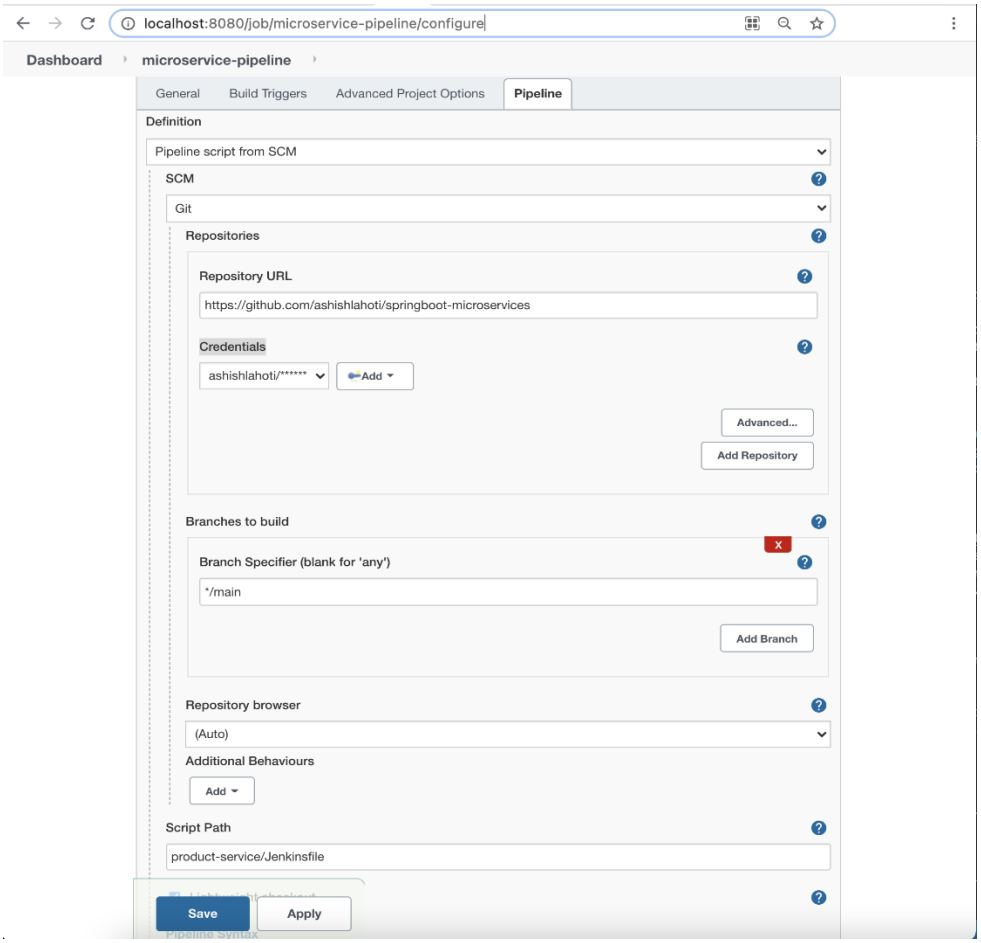
**Explanation:**

- `docker run`: The Docker command to run a container.
- `-p 8080:8080`: Maps port 8080 of the container to port 8080 on the host machine.
- `demo-app`: The name of the Docker image to run.

## 14. Steps to build Jenkins pipeline to deploy spring boot application to Docker?




1. Go to the Jenkins **Dashboard**
2. Click on **New Item** from menu
3. Enter an item name for e.g. `microservice-pipeline` and select **Pipeline** from options. Click **OK**.
4. You will see a dialogue with four tabs: **General**, **Build Triggers**, **Advance Project Options**, and **Pipeline**.
5. You can skip **General**, **Build Triggers** and **Advance Project Options** tabs. No configuration required.
6. **Pipeline** tab:
    - Choose **Definition:** `Pipeline Script from SCM`
    - **SCM:** `Git`
    - **Repositories/Repository URL:** Give repository name for e.g. `https://github.com/ashishlahoti/springboot-microservices`
    - **Repositories/Credentials:** Give username and password to access the repository
    - **Branched to build/Branch Specifier:** Give branch name from where you want jenkins to fetch Jenkinsfile for e.g. `*/main`
    - **Script path:** Give path of Jenkinsfile in the repository for e.g. `product-discovery/Jenkinsfile`
7. Click on **Save**. Congrats your pipeline is created.




[Building Docker Images using Jenkins step by step | Devops Integration Live Demo | JavaTechie]

**15. Docker Spring Boot Application?**

[Create Docker Image for Spring Boot App: Step by Step]

**16. Docker Composer Spring Boot Application?**

https://www.javaguides.net/2022/12/spring-boot-mysql-docker-compose-example.html

**17. Docker Installation steps in windows?**

[How To Install Docker on Windows 11 - Step-by-Step for Beginners (Updated 2025)]

**18. On What Circumstances Will You Lose Data Stored in a Container?**

The Data in a container can be lost whenever the container is deleted, or if docker non-persistent storage ( Ephemeral storage ) is used without proper data management. To make the data persistent , it is recommended to use Docker volumes or volume binding ( volume mounts ) are recommended.

**19. How Do You get the Number Of Containers Running, Paused, and Stopped?**

For obtaining the number of running, paused, and stopped containers in Docker you can use the command such as **`docker ps -q`** for knowing the list of running containers and **`docker ps -q -f "status=paused"`** for paused ones. Stopped containers can be counted using **`docker ps -aq -f "status=exited"`**. These commands will the provide the list of container IDs , and you have to can further process the output to get the counts programmatically like **`docker ps -q | wc -l`.**

**20. How to Start, Stop, and Kill a Container?**

To start the docker container use this command:

```
docker start < container_name >
```

To stop the docker container use this command:

```
docker stop < container_name >
```

To kill the docker container use this command:

```
docker kill < container_name >
```

**21. Can a Container Restart By Itself?**

Yes, a container itself can restart automatically by setting up the --restart option during the creation period of time. For example using **`docker run --restart`** always. This will ensure that the container restarts irrespective of its exit status.

```
docker run --restart
```

**22. How Do You Scale Docker Containers Horizontally?**

Horizontal scaling is achieved through replicating the services across multiple nodes. Tools like Docker Compose or Docker Swarm facilitate this process. For example, using **`docker-compose up --scale web=3`** command will replicates the **"web"** service to three instances, distributing the workload across them horizontally.

```
docker-compose up --scale web=3
```

**23. What Is the Difference Between Docker Restart Policies "no", "on-failure," And "always"?**

These restart policies will provide flexibility in managing the container behavior based on specific requirements. The restart policy **"no"** gives full control over restarts, **"on-failure"** handles irregular issues, and **"always"** will ensures the constant availability. Choose

the appropriate policy based over the nature and importance of that particular containerized application.

**24. Can We Use JSON Instead Of YAML While Developing a Docker-Compose File in Docker?**

Yes, Docker Compose has support for both YAML and JSON formats for defining the configuration of services. While YAML is more commonly used due to its readability and clearness , you can also use JSON as an alternative. The choice between of two will depends on personal preference on requirements of the projects. To use JSON, simply try on creating a `**docker-compose.json**` file instead of a `**docker-compose.yml**` file, and define your services in JSON format.

**25. How Will You Ensure Container 1 Runs Before Container 2 While Using docker-compose?**

In Docker Compose, the order of the services startup is determined by their dependencies. By specifying container dependencies with the **"depends_on"** key in the docker-compose.yml file, you can ensure the desired startup order.
A sample example on usage of depends_on provided here , In this even though container1 is listed firsted because of the **depends_on** key container2 will be startup and then container1 will be queue order.

```
services:
  container1:
    depends_on:
     - container2
   ...
  container2:
  ....
```

**26. Is it a Good Practice to Run Stateful Applications on Docker?**

Docker is primarily designed for the stateless applications. On using Docker volumes or persistent storage stateful applications can be runnable but it's crucial to carefully manage data persistence and backup to avoid data loss.

27. **Is it better to directly remove the container using the rm command or stop the container followed by remove container?**

Its always better to stop the container and then remove it using the remove command.

```
$ docker stop <coontainer_id>
$ docker rm -f <container_id>
```

Stopping the container and then removing it will allow sending SIG_HUP signal to recipients. This will ensure that all the containers have enough time to clean up their tasks. This method is considered a good practice, avoiding unwanted errors.

28. **Is it a good practice to run stateful applications on Docker?**

The concept behind stateful applications is that they store their data onto the local file system. You need to decide to move the application to another machine, retrieving data becomes painful. I honestly would not prefer running stateful applications on Docker.

29. **How will you monitor Docker in production?**

    Docker provides functionalities like docker stats and docker events to monitor docker in production. Docker stats provides CPU and memory usage of the container. Docker events provide information about the activities taking place in the docker daemon.